An algorithm for large scale density matrix renormalization group calculations

Garnet Kin-Lic Chan

Citation: **120**, (2004); doi: 10.1063/1.1638734
View online: http://dx.doi.org/10.1063/1.1638734
View Table of Contents: http://aip.scitation.org/toc/jcp/120/7
Published by the American Institute of Physics

An algorithm for large scale density matrix renormalization group calculations

Garnet Kin-Lic Chan
Department of Chemistry, University of Cambridge, CB2 1EW, United Kingdom

(Received 13 August 2003; accepted 12 November 2003)

We describe in detail our high-performance density matrix renormalization group (DMRG) algorithm for solving the electronic Schrödinger equation. We illustrate the linear scalability of our algorithm with calculations on up to 64 processors. The use of massively parallel machines in conjunction with our algorithm considerably extends the range of applicability of the DMRG in quantum chemistry. © *2004 American Institute of Physics.* [DOI: 10.1063/1.1638734]

INTRODUCTION

The density matrix renormalization group (DMRG) algorithm of White[1] has emerged as one of the most promising new routes to computing high-accuracy solutions of the Schrödinger equation. Fano, Ortolani, and Ziosi[2] performed the first DMRG calculations on molecules using the Pariser–Parr–Pople (PPP) Hamiltonian; this was followed shortly by the work of White and Martin[3] where the full electronic Hamiltonian was used. Since then, a number of groups[4–7] have implemented their own versions of the algorithm and today several DMRG codes for quantum chemistry are in existence.

In the arena of high-accuracy calculations, quantum chemical methods such as full configuration interaction (FCI) theory have long taken advantage of the latest advances in computing power. The dominant supercomputer platforms today are of the massively parallel, distributed memory type. Using such computers, many impressive quantum chemical calculations have been performed, most notably the recent large FCI calculations of Rossi *et al.*[8] which handled 9.6×10^9 determinants in D_{2h} symmetry.

In a recent communication,[9] we reported the exact solution of the Schrödinger equation for the water molecule, in a triple zeta, double polarization basis of 41 functions, using the DMRG method. Such a calculation, which if done with a FCI algorithm would involve an infeasibly large determinantal space (5.6×10^{11} determinants), was facilitated by our recent development of a fully parallel DMRG algorithm optimized for distributed memory architectures. The detailed description and discussion of this algorithm is the subject of the present work.

THE DMRG ALGORITHM: AN OVERVIEW

Our formulation of the DMRG algorithm is similar to that presented in our earlier work[4] with a few exceptions noted below. We shall only review the parts relevant to our current work here; for full details, we refer the reader to Ref. 4.

The DMRG algorithm consists of a set of sweeps forwards and backwards along a one-dimensional ordering of k orbitals, called a lattice. The lattice is divided into four blocks: two large blocks L and R, whose sizes vary in the course of a sweep and two small blocks B_L, B_R, of constant size, which act as a ''buffer'' in between (Fig. 1). If the sweep is going from left to right then L is the system block, which grows in size during the sweep, and R is the environment block, which decreases in size. Sweeps consists of a set of *sweep iterations* in which the system block increases in size by the number of orbitals in the adjacent B block, while the size of the environment decreases by the corresponding amount. A sweep terminates when sufficient iterations have passed for the system to span all the orbitals in the lattice, and the next sweep commences in the reverse direction. This is indicated in Fig. 2.

Each block contains a many-body space which is a subspace of the Fock space spanned by the orbitals in the block. The small block spaces $\{B_L\}$, $\{B_R\}$ are complete and contain four states, consisting of $|\ \rangle$, $|\uparrow\rangle$, $|\downarrow\rangle$, $|\uparrow\downarrow\rangle$, while the system and environment blocks spaces $\{L\}$, $\{R\}$ are restricted to be spanned by at most M many-particle basis states. The essence of the DMRG algorithm is to determine the optimal M many-particle basis states in which to represent the system and environment blocks. In our DMRG calculations, M is specified at the beginning of the calculation, and the many-particle basis states are then improved in successive sweeps until convergence is achieved.

Sweep iterations are conveniently divided into three steps.

(i) Blocking, where the system and environment are enlarged by combining with their respective neighbor blocks B_L or B_R to form a system superblock and environment superblock. The superblocks each contain $O(4M)$ many-particle states.

(ii) Solving for the wave function in the product space of the system and environment superblocks. The product space is of dimension $O(16M^2)$ and is restricted to contain the correct number of particles and other conserved quantities of our problem.

(iii) Decimation/transformation, where we transform our system representation from the superblock space of dimension $O(4M)$ to a new many-body basis of dimension M. With this decimated system block, we can start a new iteration at step (i).

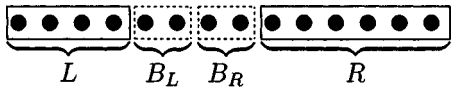

FIG. 1. A standard block configuration. B_L is to be blocked with L, and B_R with R. Each dot (or site) represents a spin–orbital.

In the DMRG, because the Slater determinant representations of the complicated M many-body basis states of the system and environment are not kept, we must store complete matrix representations of a number of intermediate operators, in order to reconstruct the Hamiltonian operator and any other operators whose properties we wish to compute. For example, if we wish to compute $a_i a_j$, $i \in L$, $j \in B_L$, we would need the tensor product of intermediate operators a_i and a_j in the $\{L\}$ and $\{B_L\}$ spaces, respectively. During blocking, we form the matrix representations of all the operators in the spaces spanned by the superblocks $\{L\}\otimes\{B_L\}$ and $\{B_R\}\otimes\{R\}$, from corresponding operators in the $\{L\}$, $\{R\}$ and $\{B_L\}$, $\{B_R\}$ spaces.

There is flexibility in the decomposition of the Hamiltonian into intermediate operators on the left and right blocks which leads to different algorithms. It is usual to construct contracted operators[10] such as $P_{ij}=\Sigma_{kl\in\text{blk}} v_{ijkl} a_k a_l$ (where $kl\in$ blk denotes all orbitals in the block). Introducing P_{ij} leads to greater storage requirements than computing $A_{ij}=a_i a_j$ alone, but the advantage is that the corresponding contribution to the interaction Hamiltonian between blocks 1 and 2, say

$$H_{\text{int}}=\sum_{ijkl} v_{ijkl} A_{ij}^{\dagger 1}\otimes A_{kl}^{2}, \tag{1}$$

which involves a four-index summation, can be computed now through the simpler two-index dot-product operation

$$H_{\text{int}}=\sum_{ij} A_{ij}^{\dagger 2}\otimes P_{ij}^{1}. \tag{2}$$

We shall refer to pairs such as $A_{ij}^{\dagger}$, P_{ij} as contraction pairs. Similarly, we have triply contracted operators such as S_i $=\Sigma_{jkl\in\text{blk}} w_{ijkl} a_j^{\dagger} a_k a_l$ which may be efficiently formed from doubly contracted operators through such terms as

$$S_i=\sum_{j\in 2} (Q_{ij}^{1}\otimes a_j^{2}+2P_{ij}^{1}\otimes a_j^{\dagger 2})+S_i^{1}\otimes \mathbf{1}^{2}+\mathbf{1}^{1}\otimes S_i^{2}. \tag{3}$$

Explicit formulas for all contracted operators in the DMRG can be found in the Appendix.

Contractions allow a trade-off between computational time and storage cost and balancing these two is an important concern when formulating a high-performance algorithm. Our approach has been to preserve the optimal asymptotic cost scaling in terms of time or memory (as a function of the number of orbitals k and DMRG states M) of the original algorithm in Refs. 4 and 3, although the relative prefactors may be adjusted as necessary depending on the computer architecture and the problem one is studying.

In our recent large-scale calculations,[9] we used the decomposition of operators listed in Table I. This differs from our earlier work[4] principally in that the doubly contracted operators P_{ij} and Q_{ij} are built on *both* L and R blocks rather than the L block alone. This step requires an additional $O(M^2k^3)$ step per sweep iteration and $O(M^2k^2)$ storage; but we have found that in a parallel algorithm, the construction of the triply contracted R_i operator is a bottleneck (see below) and our current decomposition allows S_i to be computed via Eq. (3) with $O(M^2k^2)$ effort per iteration from doubly contracted operators, rather than the $O(M^2k^3)$ time required to build S_i from uncontracted operators. To reduce storage requirements from Ref. 4, we have folded the contribution of the contracted one-index operator $\Sigma_{j\in\text{blk}} t_{ij} a_j$ into S_i. The most expensive step in blocking is then the formation of the two-index operators A_{ij}, B_{ij}, P_{ij}, Q_{ij}, which requires $O(M^2k^3)$ time per sweep iteration.

The largest spaces for which we need explicit operator matrices are the superblock spaces $\{L^{\text{super}}\}=\{L\}\otimes\{B_L\}$ and $\{R^{\text{super}}\}=\{B_R\}\otimes\{R\}$, in which operators require $O(16M^2)$ storage. Instead of storing the superblock representations of all the operators, we have the option to compute them in a "direct" fashion, i.e., to compute the explicit representations in the $O(4M)$ superblock space only when needed, such as when solving for the groundstate wave function in the Davidson procedure. The choice of which operators are to be computed in a direct fashion and which should be stored in memory is governed by the balance between the cost of storing and cost of computing the operators. These costs are intimately linked with the issue of parallelization as certain operators (such as the triply contracted operator S_i) are particularly expensive to compute in a distributed fashion and thus their superblock representations are better stored in memory.

Once we have the necessary intermediate operators (direct or otherwise) in the superblock spaces, we can solve for the wave function in the tensor product space $\{L^{\text{super}}\}\otimes\{R^{\text{super}}\}$. As described in our earlier work,[4] we use the Davidson algorithm to solve for the required eigenfunction. In common with other iterative eigenvalue algorithms, the fundamental operation is the action of the Hamiltonian on the wave function c in the tensor product space. Because of the product structure of the space and because the Hamil-

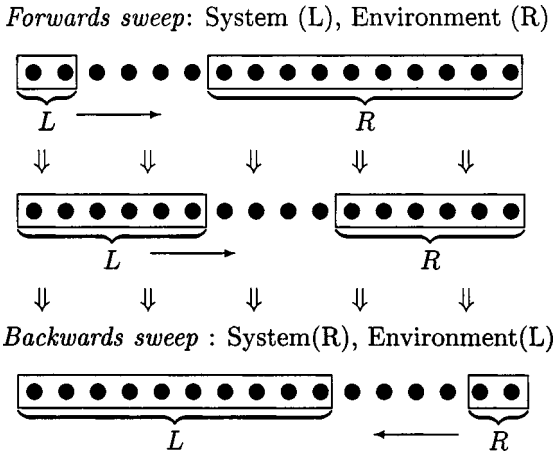

FIG. 2. The DMRG sweep algorithm. In the forwards sweep, the system block L, is grown two sites at a time. In the backwards sweep, R becomes the system block.

TABLE I. Distribution and storage of operators amongst the blocks. *r*, replicated; *d*, distributed; 0, stored only on processor 0; *D*, direct; *N*, not computed. Matrix elements, $x_{ijkl}=v_{ijkl}-v_{jikl}-v_{ijlk}+v_{jilk}$, $w_{ijkl}=v_{ijkl}-v_{jikl}$.

Operator	Definition	L	LB_L	R	B_RR	B_L, B_R
a_i	a_i	r	rD	r	rD	r
A_{ij}	a_ia_j	N	N	d	dD	r
B_{ij}	$a_i^\dagger a_j$	N	N	d	dD	r
P_{ij}	$\Sigma_{kl\in\text{blk}}v_{ijkl}a_ka_l$	d	dD	d	dD	r
Q_{ij}	$\Sigma_{kl\in\text{blk}}x_{ijkl}a_k^\dagger a_l$	d	dD	d	dD	r
S_i	$\Sigma_{jkl\in\text{blk}}w_{ijkl}a_j^\dagger a_ka_l+\Sigma_{j\in\text{blk}}t_{ij}a_j$	d	d	d	d	r
H	$\Sigma_{ij}t_{ij}a_i^\dagger a_j+\frac{1}{2}\Sigma_{ijkl}v_{ijkl}a_i^\dagger a_j^\dagger a_ka_l$	0	$0D$	0	$0D$	r

tonian decomposes into products of operators $H=\Sigma_{\text{ops}}O_LO_R$, where O_L and O_R are operators that act, respectively, on the superblock spaces $\{L^{\text{super}}\}$ and $\{R^{\text{super}}\}$ alone, this operation can be performed in two stages:

$$v_{l'r'}=H_{ll'rr'}c_{lr}=\sum_{\text{ops}}[O_L]_{ll'}[O_R]_{rr'}c_{lr} \tag{4}$$

$$U_{l'r}=\sum_{\text{ops}}c_{lr}[O_R]_{rr'} \tag{5}$$

$$v_{l'r'}=\hat{p}U_{lr'}[O_L]_{ll'}, \tag{6}$$

where l, l', r, r' denote indices of states on the left and right superblocks, respectively, and $\hat{p}$ generates the appropriate coupling element, -1 or 1.[4] The above summation is over all contraction pairs as well as such terms as $H_L\otimes\mathbf{1}$ and $\mathbf{1}\otimes H_R$. Thus the Hc operation is of a dot-product form. Each term in the summation takes $O(M^3)$ time, leading to a total time cost of $O(M^3k^2)$ (coming from contraction pairs with two indices) for each Hc operation in the Davidson algorithm. Furthermore, if O_R and O_L are operators computed in a direct fashion, we store at most five large $O(16M^2)$ matrices in memory: O_L, O_R, c, v, and the diagonal elements of H used for the Davidson procedure.

After determining the ground state wave function c, we proceed to the decimation/transformation step where the space spanned by the system superblock of size $O(4M)$ is transformed to our desired size M. As argued by White,[1] the optimal many-body basis to transform into is given by the M eigenvectors of the system reduced density matrix with largest weights. This density matrix is formed by tracing out the contributions of the environment states to the full N particle density matrix, namely

$$\Gamma_{ll'}=\sum_r c_{lr}c_{l'r}, \tag{7}$$

where l and r are the system and environment superblock states (or reversed if the sweep direction is from right to left). Thus, after c is determined, we form $\Gamma_{ll'}$, and diagonalize to determine its eigenvectors C and corresponding weights w. From these weights we pick out the M eigenvectors of Γ with largest weight and rotate all matrix representations of operators of the system superblock into this new density matrix basis. The cost of each rotation, per operator, is $O(M^3)$, which leads to $O(M^3k^2)$ cost per sweep iteration. After all the transformed operators are formed, they are saved to disk to be reused in the following sweep as operators of the environment when the roles of system and environment are interchanged.

Steps (i), (ii), and (iii) form a single sweep iteration. One sweep consists of a succession of $O(k)$ sweep iterations, where the system grows from an initially very small size (where the corresponding system space can be treated exactly) until it spans the entire lattice. After each sweep, the energies of the wave functions of all the different sweep iterations are compared and the lowest (which generally occurs near the middle of the sweep) is denoted the sweep energy. The sweeps are then carried out until convergence in the sweep energy is observed. The remaining error in the converged sweep energy, which is variational, is from the decimation to M states. M need not be held constant between sweeps: it may be increased after each sweep if higher accuracy is desired. In total, since there are $O(k)$ iterations per sweep, the cost of the DMRG algorithm per sweep is $O(M^3k^3)+O(M^2k^4)$ in time, $O(M^2k^2)$ in memory, and $O(M^2k^3)$ in disk.

WARM UP ALGORITHM

In the first sweep of the DMRG, we need to supply an approximate representation of the environment block. In many calculations, the converged DMRG energy does not depend on this initial representation. However, since we are decimating our system block states in an approximate way, it is sometimes possible in our early sweeps to perform a ''bad'' truncation, where we throw away all states of some given particle number and symmetry type. This ''quantum number,'' i.e., an eigenvalue that labels a class of states, such as spin, particle number, or symmetry irrep, may in fact be an important component of the ground state, but may not be recovered in subsequent sweeps. Our DMRG calculation will then proceed to converge on an energy which is artificially too high, as a result of the poor initialization procedure.

In our previous work we described a warm up procedure where the environment block contained Slater determinants chosen to have ''complementary'' quantum numbers to our system block.[4] In addition, we included a statistical correction to our density matrix which would, in a probabilistic way, reintroduce lost quantum numbers into our system.

In larger calculations, where the large number of particles implies a larger number of quantum numbers, we have used a simple modified decimation procedure during our first

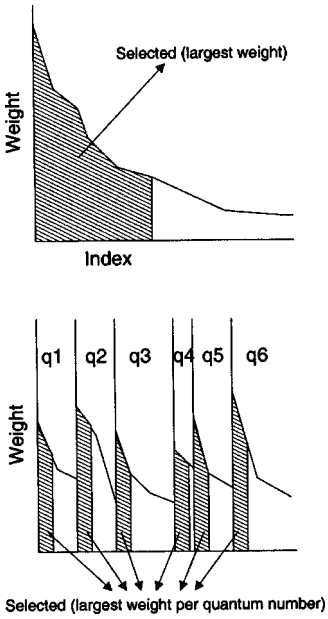

FIG. 3. Different state selection algorithms. In the upper scheme, we retain states purely by the largest overall weight in the density matrix. In the second scheme we bin the remaining states into different quantum numbers and select the first few states of largest weight per quantum number (up to a specified maximum total number of states).

few sweeps to further ensure that quantum numbers are retained. In addition to selecting states by largest overall weight in the density matrix, we additionally select a certain percentage of the states by largest weight *per quantum number* (see Fig. 3).

For example, in a warm up sweep where we might usually retain, for example, 1000 states, we retain instead 400 states using the standard criterion (largest weight in the density matrix irrespective of quantum number). Then for the remaining 600 states, we bin the states into different quantum numbers and order them, within each bin by weight. Then from every bin, we select the state of largest weight and then the state of second largest weight and so on, up to a total of 600 states (yielding 1000 states in total). Out of the 1000 states, 600 states are then distributed *evenly* over all the quantum numbers. In the following few sweeps we decrease the percentage of states that are distributed evenly across the quantum numbers, until we finally truncate based only on the 1000 states of largest weight in the density matrix and our algorithm reduces to the usual DMRG procedure. Using this simple procedure, together with using sufficient numbers of states in our calculation, we have not observed any problems with losing quantum numbers.

PARALLELIZATION

When parallelizing an algorithm, the computational and storage costs must be divided across each of the processors. The essential question is, how big should those pieces be? Our strategy with the DMRG algorithm is based on distributing the contraction pairs across the different processors (see Fig. 4). For instance, if we consider the single index operators such as S_i, we might associate S_1 and a_1, with processor 1, S_2 and a_2 with processor 2, and so on. Two

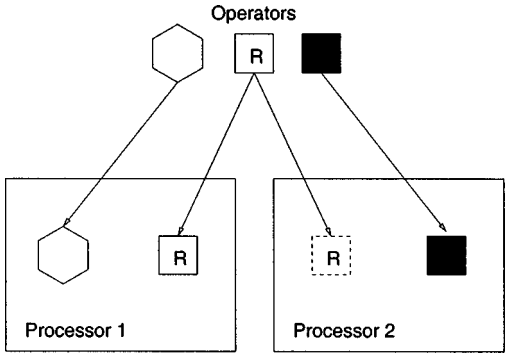

FIG. 4. Distributing operators among processors. The R label signifies a replicated operator, such as a_i. The dashed outline indicates the operator is not on its owner processor.

index operators are similarly divided up based on the pair index ij. A processor is said to *own* a set of operators and conversely, operators are stored on their owner processors. The basic scheme for parallelization then requires that processors are responsible for all steps of the sweep iteration for the operators that they own.

However, some steps require the simultaneous participation of two or more different operators. For example, during the blocking step, to form A_{15} requires knowledge of both a_1 and a_5. If these operators are not available to the processor which owns A_{15}, then communication of operators (which is $O(M^2)$ time per operator) must take place. The communication cost of our algorithm is determined primarily by the manner in which the operators are distributed.

To reduce communication costs, it is beneficial to replicate (see Fig. 4) the most frequently needed operators on all processors, so that they are available locally to every processor. The operators of the small blocks B_L and B_R take up little space and are easily replicated on all the processors. In the case of the big blocks L, R, we see that when forming the operators for the superblocks LB_L, B_RR, the operators a_i are required to build any of the other more complicated operators. In the worst case, this would lead to $O(M^2k^2)$ communication per sweep iteration, to form the two index complementary operators P_{ij}, Q_{ij} on the superblock. For this reason, we store the operators a_i for the large blocks L and R in a replicated fashion across all the processors. Our distribution of operators is given in Table I. The dominant memory cost of the DMRG algorithm is from the two index operators A_{ij}, B_{ij}, P_{ij}, Q_{ij} on the large blocks L, R, and these operators are stored in a distributed fashion, i.e., for a given index ij, A_{ij} is stored on a specific processor. The memory cost per processor is therefore reduced to $O(M^2k^2/n_p)+O(M^2k)$, where n_p is the total number of processors.

With our distribution of operators, the blocking step for a_i, A_{ij}, B_{ij}, P_{ij}, Q_{ij} is immediately parallelized with each processor performing blocking for only the operators that it owns. The replicated operators, are treated by assigning a single owner processor (usually processor 0) to such operators, which holds responsibility for steps (i), (ii), and (iii) of the sweep iteration.

For H and S_i, however, blocking still involves partial

summations over nonreplicated contraction pairs. For example, from Table I, we see that building S_i involves partial summations over contracted two index operators, of the form $\Sigma_j P^1_{ij} \otimes a^{2\dagger}_j$ and $\Sigma_j Q^1_{ij} \otimes a^2_j$. Since the two-index operators dominate the memory storage requirements, it is essential that they are not replicated over the processors.

There are a number of different possibilities for how to handle the formation of S_i and H. On computer architectures that we have tried (the IBM SP3 supercomputer at NERSC, and the Sun Fire supercomputer at the Cambridge HPCF) we have found that a strategy based on minimizing memory and compute costs at the expense of communication time, is most successful. On each processor, we build partially contracted operators which involve contractions over only local operators. For example, for S_i, we would build $S_{i\text{proc}}$ defined by (see also Appendix)

$$S_{i\text{proc}} = \sum_{j \in 2, ij \in \text{proc}} (Q^1_{ij\text{proc}} \otimes a^2_j + 2P^1_{ij\text{proc}} \otimes a^{\dagger 2}_j) + (S^1_i \otimes \mathbf{1}^2 + \mathbf{1}^1 \otimes S^2_i \quad \text{if } i \in \text{proc}). \tag{8}$$

The second line indicates that contributions to S_i that should only be included once, such as $S^1_i \otimes \mathbf{1}^2$, are built only on the owner processor of S_i to avoid double counting. Then for each S_i, we reduce over all the local two index operator contributions $S_{i\text{proc}}$ by $S_i = \Sigma^{n_p}_{\text{proc}} S_{i\text{proc}}$, with the result being accumulated on the owner processor of S_i using standard algorithms in $O(16M^2 \log n_p)$ communication and compute time. One round of accumulation is required per operator S_i, which is why we choose a nondirect algorithm for S_i (i.e., the S_i matrices for the LB_L, RB_R superblocks are stored in memory), so blocking need only be performed once per sweep iteration. H is formed in a similar manner, with contributions from (A_{ij}, P_{ij}), (B_{ij}, Q_{ij}), and (S_i, a_i) pairs computed via partial contractions as in Eq. (8), and the contribution from $(H, \mathbf{1})$ being computed only on the owner processor of H (processor 0). The largest time spent in communication comes from the accumulations performed when building S_i, which require $O(16M^2 k \log n_p)$ time. Thus the total cost per sweep iteration for step (i) (blocking) is $O(M^2 k^3 / n_p)$ compute time, with a communication time of $O(16M^2 k \log n_p)$.

Parallelization of step (ii) of the sweep iteration, the matrix-vector product $v = Hc$, is not fundamentally different from the parallelization of the blocking step for H. The matrix-vector product Hc is a generalized dot product, with the dot operation being the action of the contraction pair on c. Thus, after c is broadcasted across all processors, each processor applies the contraction pair that it owns to c independently, yielding a partial contribution

$$[v_{\text{proc}}]_{l'r'} = \sum_{lr} \hat{p} [A^{\dagger}_{ij \in \text{proc}}]_{ll'} [P_{ij \in proc}]_{rr'} c_{lr} + \cdots. \tag{9}$$

All partial contributions v_{proc} are accumulated over all processors via $v = \Sigma^{n_p}_{\text{proc}} v_{\text{proc}}$ (see Fig. 5). Each broadcast and accumulation operation takes $O(16M^2 \log n_p)$ time. Thus, the total cost for the parallelized Davidson step of the sweep iteration is $O(M^3 k^2 / n_p)$ compute and $O(16M^2 \log n_p)$ communication time.

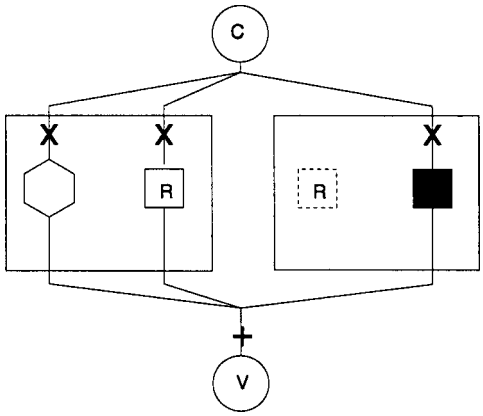

FIG. 5. Parallelization of the Davidson matrix-vector product. In the first stage, the wave function c is distributed across the processors. Multiplication $(\times)$ by all operators takes place on their owner processor, and the result is accumulated $(+)$ into v. R denotes the contribution of a replicated contraction pair, such $(H, \mathbf{1})$, which must only be included once to avoid double counting.

Step (iii) of the sweep iteration, the decimation and transformation of the operators, is also easily parallelized. After v is obtained, we form the density matrix Γ and solve for its eigenvectors using a serial algorithm on a single processor (usually processor 0). The eigenvectors of largest weight are then broadcast to all processors and each processor independently rotates the operators that it owns into this basis. As described earlier, replicated operators such as a_i are only rotated on their owner processors and therefore a synchronization must be performed after rotation, where we broadcast the transformed replicated operators from the owner to nonowner processors. This results in a communication cost (from broadcasting the a_i operators) of $O(M^2 k \log n_p)$ cost per sweep iteration (Fig. 6). The final

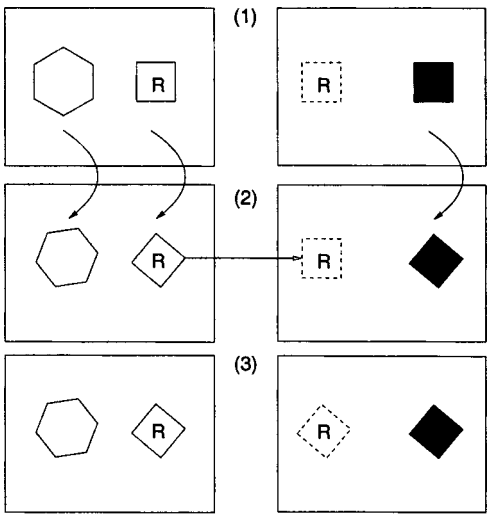

FIG. 6. Synchronization. (1) Operators are rotated except for (2) replicated operators (R) which are only rotated on their owner processors; (1) the rotated replicated operator is then broadcast from the owner processor to the other processors.

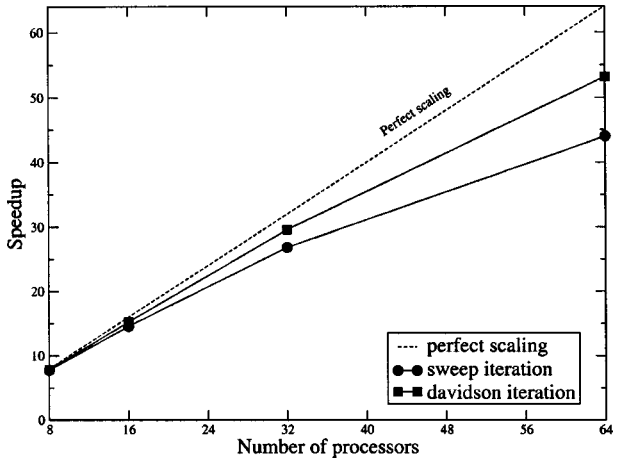

FIG. 7. Parallel speedups for the DMRG algorithm.

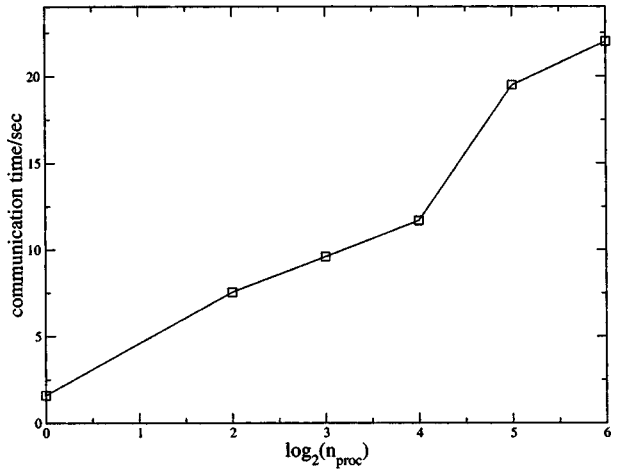

FIG. 8. Time spent in communication (during blocking) in a sweep iteration, as a function of the number of processors n_{proc}.

stage of step (iii) involves storing the operators to disk. So long as the disk access scales with the number of processors accessing the disk (for example, if each processor has a local disk, or using a high-performance file system) all saves are parallelized as each processor saves only the operators it owns to disk. The cost of step (iii) of the sweep iteration is therefore $O(M^3k^2/n_p)$ in compute time and $O(M^2k\log n_p)$ in communication time, with $O(M^2k^2)$ disk storage and access time per processor.

In summary, the total cost per sweep of our parallel algorithm, given $O(k)$ iterations per sweep is, $O(M^3k^3/n_p)+O(M^2k^4/n_p)$ compute cost, $O(16M^2k/n_p)+O(M^2k^2/n_p)$ memory cost and $O(16M^2k\log n_p)$ communication cost. Comparing this with the cost of the serial algorithm described, we see that to lowest order, all aspects of the algorithm are parallelized, at the expense of a communication cost that scales only logarithmically with the number of processors.

SCALABILITY

We tested the parallel efficiency of our algorithm with a calculation on the water molecule, correlating 10 electrons in 82 spin–orbitals in C_1 symmetry, keeping $M=1000$ states. All timings are for a single sweep iteration at a block configuration where L, R each span 40 spin–orbitals. Calculations were performed using an IBM SP3 with 375 Mhz Power3 CPUs arranged as clusters of 16-way SMP nodes, connected by a switch with a point-to-point bandwidth of 300 Mb/s.

The observed speedups are shown in Fig. 7, while the relative timings for different parts of the sweep are given in Table II. In general, good linear scaling is observed. The Davidson step forms the dominant cost of the sweep iteration and for this step we achieve typically a speedup of 56 using 64 processors (88% of the theoretical maximum). The total speedup for an entire sweep iteration is a little lower, with a speedup of 45 being achieved with 64 processors (70% of the theoretical maximum). As observed from Table II this inefficiency is almost entirely due to the increasing proportion of time spent in the serial diagonalization of the density matrix in the decimation step. While parallel diagonalization routines may be incorporated into our algorithm easily, the small matrix sizes involved [O(100×100)] will probably not allow large speedups through parallelization. However, this is not a serious barrier in principle, as we observe that diagonalization time scales like $O(M^3)$ while the Davidson algorithm time scales like $O(M^3k^2/n_p)$ per sweep iteration and thus the observed inefficiency is an artifact of the relatively small number of orbitals used in our calculation. As we contemplate calculations on systems with hundreds of orbitals, we can expect better scalability of our algorithm on larger numbers of processors.

TABLE II. Relative amounts of time (%) spent in the different steps of a sweep iteration, *a*, blocking; *b*, Davidson iteration; *c*, density matrix diagonalization; *d*, rotation of operators.

n_{proc}	*a*	*b*	*c*	*d*
1	0.0	99.0	0.4	0.6
4	0.1	98.6	0.8	0.6
16	0.5	95.2	3.8	0.6
64	2.4	81.5	15.3	0.6

Almost all the communication time of the algorithm is spent in the blocking step, during the formation of the S_i operators. For the case studied, communication costs are quite small (see Table II). Our formulated algorithm predicts a logarithmic scaling of the communication cost with the number of processors, and this is indeed observed in Fig. 8.

CONCLUSIONS

In this work we have presented a high-performance density matrix renormalization group (DMRG) algorithm. We have formulated our algorithm so that all compute, memory, and disk access costs scale linearly with the number of processors. We have observed good near-linear speedups in calculations involving up to 64 processors.

Although high accuracy DMRG calculations in quantum chemistry are expensive, the use of massively parallel machines together with our algorithm greatly increases the range of systems that we can currently treat. While we have

focused here on the popular distributed memory architecture, we note that our algorithm may also be used with little modification on shared memory machines.

Given the multireference nature of the DMRG, we now have the ability to accurately describe active spaces considerably larger than can be treated with any other method. In recent work,[9] we have treated active spaces with more than 40 orbitals essentially exactly. Further large-scale calculations are now under way.

ACKNOWLEDGMENTS

We thank the Cambridge HPCF for a grant of computer time. We would also like to thank M. Head-Gordon for support during the early part of this work.

APPENDIX: EXPLICIT FORMULAS

The formulas are given in terms of the operation $\otimes$. For blocking, $\otimes$ is a tensor product in the sense

$$\langle s_1 s_2|X_1 \otimes Y_2|s_1' s_2'\rangle = \hat{p}\langle s_1|X_1|s_1'\rangle\langle s_2|Y_2|s_2'\rangle, \tag{A1}$$

where 1 and 2 refer to the large and small block, respectively. and $\hat{p}$ generates the coupling coefficient ± 1. It is convenient to further define a symmetrized operator, $\tilde{\otimes}$, where

$$\tilde{\otimes}[X,Y] = X^1 \otimes Y^2 + Y^1 \otimes X^2, \tag{A2}$$

$$\tilde{\otimes}[X] = \tilde{\otimes}[X, \mathbf{1}]. \tag{A3}$$

For simplicity, we do not indicate the block involved if it is obvious from the expression, e.g., in $a_i \otimes \mathbf{1}$, a_i belongs to the block containing index i while $\mathbf{1}$ is the unit operator on the other block.

The expressions follow:

$$a_i: \ a_i \otimes \mathbf{1}, \tag{A4}$$

$$A_{ij}: \ A_{ij} \otimes \mathbf{1} \ \text{ or } \ a_i \otimes a_j, \tag{A5}$$

$$B_{ij}: \ B_{ij} \otimes \mathbf{1} \ \text{ or } \ a_i^\dagger \otimes a_j, \tag{A6}$$

$$P_{ij}: \ \tilde{\otimes}[P_{ij}] + \sum_{kl} v_{ijkl} a_k^1 \otimes a_l^2, \tag{A7}$$

$$Q_{ij}: \ \tilde{\otimes}[Q_{ij}] + \sum_{kl} x_{ijkl}(a_k^{\dagger 1} \otimes a_l^2 + a_l^{\dagger 2} \otimes a_k^1), \tag{A8}$$

$$S_i: \ \tilde{\otimes}[S_i] + \sum_{j \in 2} (2a_j^{\dagger 2} \otimes P_{ij}^1 + a_j^2 \otimes Q_{ij}^1), \tag{A9}$$

$$\begin{aligned} H: \ \tilde{\otimes}[H] &+ \frac{1}{2}\sum_i (\tilde{\otimes}[a_i^\dagger, S_i] + \tilde{\otimes}[S_i^\dagger, a_i]) \\ &+ \frac{1}{2}\sum_{ij \in 2} (P_{ij}^{\dagger 1} \otimes A_{ij}^2 + A_{ij}^{\dagger 2} \otimes P_{ij}^1) \\ &+ \frac{1}{2}\sum_{ij \in 2} (Q_{ij}^{\dagger 1} \otimes B_{ij}^2 + B_{ij}^{\dagger 2} \otimes Q_{ij}^1). \end{aligned} \tag{A10}$$

The expressions for the intermediates $S_{i\text{proc}}$ and H_{proc} used in the parallel algorithm are obtained from Eqs. (A9) and (A10) by restricting all operators on the rhs to only those owned by proc. In particular, we note that only the owner processor of i computes the contribution $\tilde{\otimes}[S_i]$ to $S_{i\text{proc}}$, and only processor 0 computes the contribution $\tilde{\otimes}[H]$ to H_{proc}.

The matrix vector product Hc for the Davidson iteration is generated by precisely the same formula Eq. (A10), when block labels 1, 2, are replaced by superblock labels R, L, and we redefine $\otimes$ such that $[A \otimes B]_{l'r'} = \hat{p}\Sigma_{lr} A_{ll'} B_{rr'} c_{lr}$.

[1] S. R. White, Phys. Rev. Lett. **69**, 2863 (1992).
[2] G. Fano, F. Ortolani, and L. Ziosi, J. Chem. Phys. **108**, 9246 (1998).
[3] S. R. White and R. L. Martin, J. Chem. Phys. **110**, 4127 (1999).
[4] G. K.-L. Chan and M. Head-Gordon, J. Chem. Phys. **116**, 4462 (2002).
[5] S. Daul, I. Ciofini, C. Daul, and S. R. White, Int. J. Quantum Chem. **79**, 331 (2000).
[6] O. Legeza, J. Roder, and B. A. Hess, Phys. Rev. B **67**, 125114 (2003).
[7] A. O. Mitrushenkov, G. Fano, F. Ortolani, R. Linguerri, and P. Palmieri, J. Chem. Phys. **115**, 6815 (2001).
[8] E. Rossi, G. L. Bendazzoli, S. Evangelisti, and D. Maynau, Chem. Phys. Lett. **310**, 530 (1999).
[9] G. K.-L. Chan and M. Head-Gordon, J. Chem. Phys. **118**, 8551 (2003).
[10] T. Xiang, Phys. Rev. B **53**, 10445 (1996).